COMMUNICATION

7(18), November 1, 2013

ISSN 2278–5469 EISSN 2278–5450

Discovery

Distributed file system components and characteristics

Kreetika Chhabra, Mohit Mehndiretta, Harpreet Singh

Dronacharya College of Engineering, Khentawas, Farukhnagar, Gurgaon, India

Publication History
Received: 08 September 2013
Accepted: 26 October 2013
Published: 1 November 2013

Citation
Kreetika Chhabra, Mohit Mehndiretta, Harpreet Singh. Distributed File System components and characteristics. *Discovery*, 2013, 7(18), 23-27

ABSTRACT
The distributed file systems area is a broad research area. In recent years there has been an explosion of interest using cluster of computing and shared nothing computers. In this paper we have described the design, implementation and features of Distributed File System (DFS). A dynamic distributed metadata management adapts to perform a wide range of general purpose and scientific computing file system workloads. There has been various DFS such as apache DFS, Google DFS, Linux DFS, Coda DFS, etc. which has made it a great area of research. By distributing storage and computation across many servers, the resource can grow with demand while remaining economical at every size.

Keywords: Distributed file system, characteristics of distributed file system, components of distributed file system

1. INTRODUCTION

This document describes the functionality and statistics related to a distributed system. The purpose of distributed system is to allow users of physically distributed computers to share data and storage resources by using a common file system. A DFS is implemented as a part of the operating system of each of the connected computers. The DFS is a file system, whose clients, servers and storage devices are dispersed among the machines of a Distributed System. A DFS is a distributed implementation of classical time sharing model of a file system, where multiple users share files and storage resources. The purpose of a DFS is to support the same kind of sharing when users are physically dispersed in distributed system. The need to share resources in a computer system arises due to economics or the nature of some applications. In such cases, it is necessary to facilitate sharing long term storage devices and their data. For this purpose, we discuss Distributed File System (DFS) as the means of sharing storage space and data.

File System is a method for storing and organizing computer files and the data they contain to make it easy to find and access them. Distributed File System is a network file system where a single file system can be distributed across several physical computer nodes. Separate nodes have direct access to only a part of the entire file system, in contrast to shared disk file systems where all nodes have uniform direct access to the entire storage. The characteristics of File System are:

- most files are small—transfer files rather than disk blocks?
- reading more common than writing
- most access is sequential
- most of the files have a short lifetime—lots of applications generate temporary files (such as a compiler)
- file sharing (involving writes) is unusual—argues for client caching
- processes use few files
- files can be divided into classes—handle ”system” files and “user” files differently.

A DFS is a distributed implementation of the classical timesharing model of a file system, where multiple users share files and storage resources. The UNIX timesharing file system is usually regarded as the model. The purpose of DFS is to allow users of physically distributed computers to share data and storage resources by using a common file system. A typical configuration for a DFS is a collection of workstations and mainframes connected by a LAN. A DFS is implemented as part of the operating system of each of the connected computers. The distributed file systems area is a broad research area. The more recent research deals with the need of improvements in availability of the systems and the integration/adaptation to the web scenario. Tanenbaum defines a distributed system as a “collection of independent computers that appear to the users of the system as a single computer.” There are two essential points in this definition. The first is the use of the word independent. This means that, architecturally, the machines are capable of operating independently. The other thing is that the software enables this set of connected machines to appear as a single computer to the users of the system. This is known as the single system image and is a main goal in designing distributed systems that are easy to maintain and operate.

2. STRUCTURES AND IMPLEMENTATION OF DFS

To explain the structure of a DFS, we need to define:

1.Service
2.Server, and
3.Client.

A service is a software entity running on one or more machines and providing a particular type of function to a prior unknown clients. A server is the service software running on a single machine. A client is a process that can invoke a service using a set of operations that form its client interface. Sometimes, a lower level interface is defined for the actual cross-machine interaction. When the need arises, We refer to this interface as the inter-machine interface. Clients implement interfaces suitable for higher level applications or direct access by humans. Using the above terminology, we say a file system provides file services to clients. A client interface for a file service is formed by a set of file operations. The most primitive operations are:

- Create a file,
- Delete a file,
- Read from a file, and
- Write to a file.

A method of storing and accessing files is based on client/server architecture. In a distributed file system, one or more central servers store files that can be accessed, with proper authorization rights, by any number of remote clients in the network. Much like an operating systems organizes files in hierarchal file management system, the distributed systems uses a uniform naming convention and a mapping scheme to keep track of where files are located. When the client device retrieves a file from the server, the first file appears as a normal file on the client machine, and the user is able to work with the file in the same ways as a normal file on the client machine, and the user is able to work the file in the same ways as if it were stored locally on the workstation.

2.1. Mapping of logical and physical folders

The problems which arises due to normal file system such as file sharing and file replication are solved by Distributed File System by Mapping of Logical and Physical folders of the file.

File and File Systems

File name -Mapping symbolic name to a unique file id (ufid or file handle) which is the function of directory service.

File attributes -ownership, type, size, timestamp, access authorization information. Data Units - Flat / Hierarchical Structure

File Access - sequential, direct, indexed-sequential

2.2. Architecture of Distributed System

1. UNIX file system layer - does normal open / read / etc. commands.

2. Virtual file system (VFS) layer -

Gives clean layer between user and file system.

a. Acts as deflection point by using global v nodes.
b. Understands the difference between local and remote names.

c. Keeps in memory information about what should be deflected (mounted directories) and how to get to these remote directories.

3. **System call interface layer -**
   a. Presents sanitized validated requests in a uniform way to the VFS.
   b. Break the complete pathname into components.
   c. For each component, do an NFS lookup using the component name + directory v node.
   d. After a mount point is reached, each component piece will cause a server access.
   e. Can't hand the whole operation to server since the client may have a second mount on a subsidiary directory (amount on a mount).
   f. A directory name cache on the client speeds up lookups.

3. DISTRIBUTED FILE SYSTEM FEATURES

The Distributed File System (DFS) provides several important features, described in the following sections.

3.1. Transparency Properties

Login Transparency: User can log in at any host with uniform login procedure and perceive a uniform view of the file system.

Access Transparency: Client process on hosts has uniform mechanism to access all files in system regardless of files are on local/remote host.

Location Transparency: The names of the files do not reveal their physical location.

Concurrency Transparency: An update to a file should not have effect on the correct execution of other process that is concurrently sharing a file

Replication Transparency: Files may be replicated to provide redundancy for availability and also to permit concurrent access for efficiency.

3.2. Fault Tolerance

Fault Tolerance is a design that enables a system to continue operation, possibly at a reduced level, rather than failing completely, when some part of the system fails. Fault tolerance is an approach by which reliability of a computer system can be increased beyond what can be achieved by traditional methods. While hardware supported fault tolerance has been well-documented, the newer, software supported fault tolerance techniques have remained scattered throughout the literature. Comprehensive and self-contained, this book organizes that body of knowledge with a focus on fault tolerance in distributed systems. (A single process case is treated as a special case of distributed systems.)

3.3. Scalability

Scalability is the ability of a system, network, or process to handle a growing amount of work in a capable manner or its ability to be enlarged to accommodate that growth. For example, it can refer to the capability of a system to increase total throughput under an increased load when resources (typically hardware) are added. An analogous meaning is implied when the word is used in an economic context, where scalability of a company implies that the underlying business model offers the potential for economic growth within the company. When it comes to any large distributed system, size is just one aspect of scale that needs to be considered. Just as important is the effort required to increase capacity to handle greater amounts of load, commonly referred to as the scalability of the system. Scalability can refer to many different parameters of the system: how much additional traffic can it handle, how easy is it to add more storage capacity, or even how many more transactions can be processed. A system, whose performance improves after adding hardware, proportionally to the capacity added, is said to be a scalable system. A routing protocol is considered scalable with respect to network size, if the size of the necessary routing table on each node grows as O(log N), where N is the number of nodes in the network.

3.4. Security

Distributed Systems Security provides a holistic insight into current security issues, processes, and solutions, and maps out future directions in the context of todays distributed systems. The rapid development and increasing complexity of computer systems and communication networks coupled with the proliferation of services and applications in both Internet-based and ad-hoc based environments have brought network and system security issues to the fore.

This insight is elucidated by modeling of modern day distributed systems using a four-tier logical model –host layer, infrastructure layer, application layer, and service layer (bottom to top). The authors provide an in-depth coverage of security threats and issues across these tiers. Additionally the authors describe the approaches required for efficient security engineering, alongside exploring how existing solutions can be leveraged or enhanced to proactively meet the dynamic needs of security for the next-generation distributed systems. The practical issues thereof are reinforced via practical case studies.

Distributed Systems Security:

- Presents an overview of distributed systems security issues, including threats, trends, standards and solutions.
- Discusses threats and vulnerabilities in different layers namely the host, infrastructure, application,
- and service layer to provide a holistic and practical, contemporary view of enterprise architectures.

4. SEMANTICS OF FILE SHARING

The three main semantics of file sharing are as follows:

- Unix semantics – every operation on a file is instantly visible to all processes.
- session semantics – no changes are visible to other processes until the file is closed.
- immutable files – files cannot be changed (new versions must be created)

The analysis of file sharing semantics is that of understanding how files behave. For instance, on most systems, if a read follows a write, the read of that location will return the values just written. If two writes occur in succession, the following read will return the results of the last write. File systems that behave this way are said to observe sequential semantics. Sequential semantics can be achieved in a distributed system if there is only one server and clients do not cache data. This can cause performance problems since clients will be going to the server for every file operation (such as single-byte reads). The performance problems can be alleviated with client caching. However, now if the client modifies its cache and another client reads data from the server, it will get obsolete data. Sequential semantics no longer hold. One solution is to make all the writes write-through to the server. This is inefficient and does not solve the problem of clients having invalid copies in their cache. To solve this, the server would have to notify all clients holding copies of the data. Another solution is to relax the semantics. We will simply tell the users that things do not work the same way on the distributed file system as they did on the local file system. The new rule can be "changes to an open file are initially visible only to the process (or machine) that modified it." These are known as session semantics.

That is, a file cannot be open for modification, only for reading or creating. If we need to modify a file, we'll create a completely new file under the old name. Immutable files are an aid to replication but they do not help with changes to the file's contents (or, more precisely, that the old file is obsolete because a new one with modified contents succeeded it). We still have to contend with the issue that there may be another process reading the old file. It's possible to detect that a file has changed and start failing requests from other processes.

A final alternative is to use atomic transactions. To access a file or a group of files, a process first executes a begin transaction primitive to signal that all future operations will be executed indivisibly. When the work is completed, an end transaction primitive is executed. If two or more transactions start at the same time, the system ensures that the end result is as if they were run in some sequential order. All changes have an all or nothing property.

5. CACHING

Reduce network traffic by retaining recently accessed disk blocks in a cache, so that repeated accesses to the same information can be handled locally. In the distributed environment, different activities occur in concurrent fashion. Usually, common resources like the underlying network, Web/application servers, database servers, and cache servers are shared by many clients. Distributing the computing load is the hallmark of distributed systems. Resource sharing and allocation is a major challenge in designing distributed architecture. For example, consider a Web-based database-driven business application. The Web server and the database server are hammered with client requests. Caching, load-balancing, clustering, pooling, and time-sharing strategies improve the system performance and availability. The performance of a caching system depends on the underlying caching data structure, cache eviction strategy, and cache utilization policy. To ensure reasonable performance of a file system, some form of caching is needed. In a local file system the rationale for caching is to reduce disk I/O. In a distributed file system (DFS) the rationale is to reduce both network traffic and disk I/O. In a DFS the client caches can be located either in the primary memory or on a disk. The server will always keep a cache in primary memory in the same way as in a local file system. The block size of the cache in a DFS can vary from the size of a disk block to an entire file.

5.1. Cache Update Policy

The policy used to write modified data back to the server's master copy has a critical effect on the system's performance and reliability. Update policies:

• Write-through

The simplest and most reliable strategy. Write operations must wait until the data is written to the server. The effect is that the cache is only used for read operations.

• Delayed write

Modification are written to the cache and then written to the server at a later time. Write operations becomes quicker and if data are overwritten before they are sent to the server only the last update need to be written to the server.

• Write-on-close

All the time the file is open, the local cache is used. Only when the file is closed, data is written to the file server. For files that are open for long time periods and frequently modified, this gives better performance than delayed write. Used by the Andrew file system.

5.2. Remote Services

When a client needs service from a server on another machine, messages need to be sent to the server demanding the service. The server sends back a message with the requested data. A common way to achieve this is Remote Procedure Call (RPC). The idea is that an RPC should look like a normal subroutine call to the client. Another possibility is to use sockets directly. Sockets used in the File system code however, have a few disadvantages:

1. Sockets may not be available in all systems
2. Making a connection using sockets requires knowledge of socket names. This is a type of system configuration data that should not be compiled into file system code.

Comparison between Caching and Remote Service

- Many remote accesses can be handled by a local cache. There's a great deal of locality of reference in file accesses. Servers can be accessed only occasionally rather than for each access.
- Caching causes data to be moved in a few big chunks rather than in many smaller pieces; this leads to considerable efficiency for the network.
- Disk accesses can be better optimized on the server if it's understood that requests are always for large contiguous chunks.
- Cache consistency is the major problem with caching. When there are infrequent writes, caching is a win. In environments with many writes, the work required to maintain consistency overwhelms caching advantages.
- Caching works best on machines with considerable local store - either local disks or large memories. With neither of these, use remote-service.
- Caching requires a whole separate mechanism to support acquiring and storage of large amounts of data. Remote service merely does what's required for each call. As such, caching introduces an extra layer and mechanism and is more complicated than remote service.

6. CONCLUSION

This research paper underlines the requirement and various features of a Distributed File System. The problem of File System is solved by the DFS. The purpose of DFS is to allow users of physically distributed computers to share data and storage resources by using a common file system. The DFS do the mapping of physical and logical folders. By which we can store and share files in an efficient manner. The features of DFS underlines its advantages and efficiency that how this system overcomes the problems and drawbacks of other file systems. The transparency property underlines various parameters, fault tolerance, scalability; security etc. describes the effectualness of the DFS system. The semantics of file sharing undergoes with three attributes as UNIX semantics, session semantics and immutable files. The caching is very important in DFS, which says that the client caches can be located either in the primary memory or on a disk. The server will always keep a cache in primary memory in the same way as in a local file system. The block size of the cache in a DFS can vary from the size of a disk block to an entire file. It is used for load-balancing, clustering, pooling, and time-sharing strategies improve the system performance and availability. Then the remote access explains how the data is remotely accessed and used. The comparison between remote access and caching determines which is more versatile and why, we clearly observe that caching is much better and efficient then remote access system as caching overcomes all the problem of remote services. Because the dependence on distributed file systems increases, the availability becomes a serious concern. For example, the AFS can cache entire files on local disks, which allow read access avoid server interaction. But there is write dependence yet. Thus DFS is sensitive to failures of servers and the network even though the dependence is minimal.

REFERENCE

1. Chow R, Johnson T. Distributed Operating Systems & Algorithms, 1997
2. Eliezer Levy, Abraham Silberschatz. December 1990 Computing Surveys (CSUR), 22(4), Distributed file systems: concepts and examples
3. Howard JH, Kazar ML, Menees SG, Nichols DA, Satyanarayanan M, Sidebotham RN, West MJ. Scale and Performance in a Distributed File System. ACM Transactions on Computer Systems, 1988, 6(1), 1988
4. Open Software Foundation, Inc. File Systems in a Distributed Computing Environment. White Paper. 1991
5. Randy chow. Distributed operating systems & Algorithms, 1997
6. Thanh TD. Mohan S, Choi E, SangBum Kim, Pilsung Kim. Networked Computing and Advanced Information Management. A Taxonomy and Survey on Distributed File Systems, 2008
7. Walter B, Popek G, English R, Kline C, Thiel G. The Locus Distributed Operating System. Proceeding of the Ninth ACM Symposium on Operating Systems Principles, Breton Woods. Oct 1983